Timing Requirement Description Diagrams for Real-Time System Verification

B. Fontan[1, 2], P. de Saqui-Sannes[1, 2], L. Apvrille[3]

1: LAAS-CNRS
7 Avenue du Colonel Roche, 31077 Toulouse Cedex 04, France

2: University of Toulouse
ISAE
10 avenue Edouard Belin - BP 54032 - 31055 Toulouse cedex 4

3: institut TELECOM, TELECOM ParisTech
2229 route des Crêtes, B.P. 193, 06904 Sophia-Antipolis Cedex, France

bfontan@isae.fr, pdss@isae.fr, ludovic.apvrille@telecom-paris.fr

Abstract: TURTLE is a real-time UML profile introduced a few years ago to address the analysis, design and deployment of time-constrained systems. The profile has a formal semantics. Further, it is supported by an open source toolkit: TTool. The latter enables formal verification of TURTLE models without specific knowledge of mathematical notations or formal languages. This paper proposes to extend TURTLE to cover the requirement capture phase, to check a model against formally expressed temporal requirements, and to achieve temporal requirement traceability. TURTLE is extended with SysML requirement diagrams. Non-formal and formal requirements are both handled. Timing Requirement Description Diagrams are introduced to formally express temporal requirements. TRDDs are based on UML Timing Diagrams. A Hybrid Power Management Unit of a Hybrid Vehicle serves as example.

Keywords: Methodology, UML, SysML, Temporal Requirement, Formal Verification.

1. Introduction

TURTLE [3] [4] is a real-time UML [16] profile supported by TTool (TURTLE toolkit [19]). The latter enables formal verification of real-time systems models. In particular, verification may be guided by observers.

The profile and TTool have recently been extended to support SysML [18] requirement diagrams. TURTLE requirement diagrams may contain informal requirements expressed in natural language. They may also include temporal requirements expressed in a chronogram style, using a Timing Requirement Description Diagram. A TRDD serves as starting point for automated synthesis of observers.

The paper is organized as follows. Section 2 presents the TURTLE profile. It also introduces a methodology for the design of real-time embedded systems, relying on TURTLE and its SysML extensions. Section 3 introduces requirement description diagrams and TRDDs. Metamodels are introduced. Also, TRDDs' expression power is discussed. Section 4 applies TURTLE to a Hybrid Power Management Unit of a Hybrid Vehicle. Section 5 concludes the paper.

2. TURTLE

2.1 Overview of the TURTLE profile

TURTLE (Timed UML and RT-LOTOS Environment) is a SysML/UML profile for real-time system analysis and design [3] [4] [20]. The profile has a formal semantics expressed by translation to RT-LOTOS [8]. It is implemented by TTool [19], an open source toolkit interfaced with two formal verification tools: RTL [17] and the BCG Tool of the CADP Toolkit [6] (the use of BCG is not addressed in this paper). Formal verification works as follows: TTool transforms a TURTLE model into an RT-LOTOS specification, and the latter's reachability graph is generated by RTL.

Formal verification may be applied to the two groups of UML diagrams customized by TURTLE[1]: (1) analysis diagrams (interaction overview and sequence diagrams), and (2) design diagrams (class and activity diagrams).

TURTLE diagrams may be edited using TTool. As shown by Figure 1, TTool translates all the diagrams into TIF, a TURTLE Intermediate Format expressed in native TURTLE [4]. TIF is made up of "basic" design diagrams. TIF serves as starting point to generate either an RT-LOTOS specification or executable Java code. Java code generation is out of scope of the paper.

[1] [4] also presents Deployment Diagrams (not addressed in this paper).

Figure 1 shows that observers are automatically generated from requirement diagrams. Observers are translated into TIF and connected to the TIF form of relevant class and activity diagrams of the system's model.

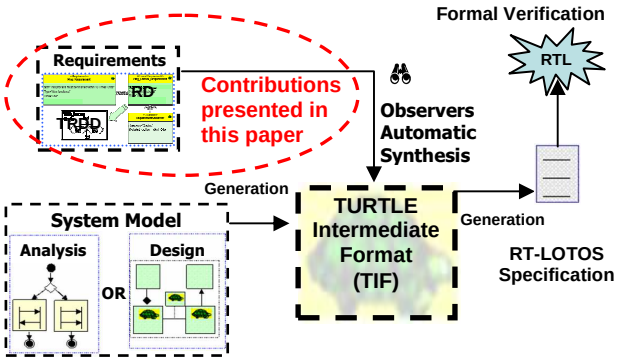

Figure 1: Main functions implemented by the TURTLE toolkit

2.2 The TURTLE Methodology

This section introduces the four-step methodology depicted by Figure 2. A previous methodology was presented in [4]; it cover steps (2) (3) and (4) (see on Figure 2, the dashed rectangle labeled by "previous methodology"). This paper proposes an enhanced methodology whose advantage is twofold. In (1), non-functional temporal requirements are depicted in a chronogram style by using a TRDD (Timing Requirement Description Diagram). Second, the TRDD serves as starting point for generating an observer in charge of guiding formal verification in step (3). Note that temporal requirement traceability is also of prime concern.

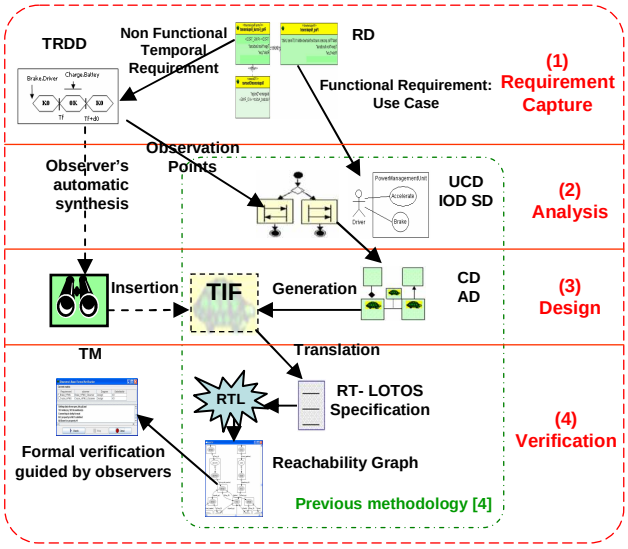

Figure 2: The TURTLE Methodology

Glossary:

RD	Requirement Diagram
TRDD	Timing Requirement Description Diagram
TM	Traceability Matrix
UCD	Uses Cases Diagram
SD	Sequence Diagram
IOD	Interaction Overview Diagram
CD	Class Diagram
AD	Activity Diagram
TIF	TURTLE Intermediate Format

The requirement capture phase (step (1)) starts with a requirement diagram (RD) definition. Each node in the RD defines one requirement in plain text, which means that the requirement in question is informal (in the sense that it is written in, e.g., English and not using a language whose syntax and semantics are formally defined). Both functional and non functional temporal requirements (TR) may be respectively expressed by use cases and TRDDs (Timing Requirement Description Diagram).

The two dashed lines in Figure 2, between steps (1) and (3), indicate that observers may be generated from TRDDs to be associated with design diagrams (CD and AD). Automatic synthesis algorithms and metamodels are described in [20].

Also in (4), a traceability matrix is automatically generated from the results collected by the observers. Verification implements a reachability analysis approach. The output is a graph (RG) that can be minimized into a quotient automaton (QA) [14]. To check whether a given desirable property holds or not, we use either the RG or the QA.

In section 4, this methodology is exemplified over a Hybrid Power Management Unit of a Hybrid Vehicle.

3. Temporal Requirement Description Language

3.1 Related Work

This section surveys various modelling techniques that might have been used to extend TURTLE with a requirement description language. The objective is to present the rationale behind the definition of TURTLE's requirement diagrams, including TRDDs.

Beyond the support of requirement capture, SysML offers system engineers a UML-based notation which is less software centric than UML 2.1 [16]. For instance, [22] proposes an extended SysML with bond graphs. The extended notation enables description of energy flows between mechanical blocks located inside one system. Unlike [22], TURTLE reuse SysML block diagrams and ignores the functional design style inherent to SysML blocks.

TURTLE requirement diagrams differ from SysML ones for they allow one to formally express requirements and to associate them with verification results.

In KAOS (Keep All Objective Satisfied [13]) requirements are expressed by means of logic formulas written in RT-LTL (Real Time Linear Temporal Logic). KAOS also includes a method for goal driven requirement elaboration. The KAOS tool Objectiver [15] enables analysts to elicit and specify requirements in a systematic way and to achieve traceability from requirements to goals. The interest of the KAOS methodology is to formalize and trace functional and non-functional requirements (including security, safety, accuracy, cost, performance) throughout the design cycle. In this paper, we also link (temporal) requirements to our formalism and we integrate requirement capture and requirement traceability.

Scenario based modelling techniques are also candidates for temporal requirement description. The verification process consists in matching [5] scenarios and the model of the system. For instance, Timed Use Case Maps [11] (see TUCM in table 1) describe Use Cases Interactions including absolute time with a master clock and relative time constraints (Duration, Timer). Also, Visual Timed events Scenario [5] (see VTS in table 1) represent events interactions. An event represents an action which potentially occurs inside the system. VTS includes time representation. It may express partial orders and relative time constraints between events. Finally, Live Sequence Charts [9] (LSC in table 1) extend Messages Sequence Charts (MSC) to represent scenarios. LSC enable distinction between possible and necessary scenarios.

Name	TUCM	VTS	LSC
Reference	[11]	[5]	[9]
Formal Language	Clocked Transition Systems	Timed Computation Tree Logic	Bücchi Automata
Verification type	Model Checking	Model Checking (UPPAAL/Kronos)	Model Checking

Table 1: Scenario-based visual languages with formal semantics

The scenario-based description languages discussed so far have a formal semantics, and so have TRDDs. TRDDs reuse the concept of observation points introduced in VTS. Nevertheless, TRDDs do not implement a scenario paradigm, which seems appropriate for the analysis phase, but not the requirement capture one.

To reduce the gap between requirement capture and formalization, temporal requirements might also be represented using Timing Diagrams. The latter make it possible to represent temporal requirements in an easy to read and formal way. The formalism used by the ICOS toolbox [10] is similar to timing diagrams. Real Time Symbolic Timing Diagrams (RT-STD in table 2) are applied to SoC design. Regular Timing Diagrams [2] (see RTD in table 2) improve the situation: they enable representation of partial order between diagrams.

Name	RT-STD	RTD	TRDD
Reference	[10]	[2]	This paper
Formal Language	Bücchi Automata	Symbolic Values	RT-LOTOS
Type of verification	Model Checking	Model Checking	Observers

Table 2: Visual Languages based on Timing Diagrams

Overall, we favor the timing diagram paradigm for its main concepts may be reused and adapted to express temporal requirements. Accordingly, TRDDs are based on timing diagrams.

3.2 Requirement Diagram (RD)

A SysML requirement is a test case [18] stereotyped by <<requirement>> and characterized by four attributes: (1) an *identifier*; (2) a *text* (an informal description of the requirement); (3) a *type* (“functional” or “non-functional”); (4) a *risk* level (“high” or “low”) depending on whether the requirement is strong or weak, respectively.

The TURTLE requirement diagram in Figure 3 includes an informal requirement and a formal one. Both address the same system constraint: “*the process must be completed within 10 time units*”.

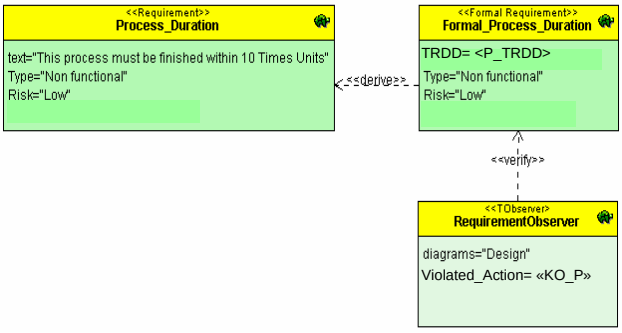

Figure 3: Example of a TURTLE Requirement Diagram

A Requirement Diagram may also describe a requirement refinement, a derivation, and a verification. In Figure 3, an informal requirement (stereotyped by <<Requirement>>) is derived (cf. the dependency relation stereotyped by <<derive>>) into a formal requirement (stereotyped by <<Formal Requirement>>). The latter is to be verified using an observer (stereotyped by <<TObserver>>). Thus, the “Formal Requirement” serves as starting point for formal verification. The *text* in the informal requirement is replaced by a Timing Requirement

Description Diagram (TRDD) in the formal requirement (see section 3.3).

Formal requirements such as the one depicted in Figure 3 serve as starting point to generate observers intended to guide verification. As shown by Figure 3, an observer contains two attributes. First, the *diagrams* field indicates the diagram for which observer are meant to be generated. It may be either an analysis or a design diagram. The second field, named *Violated_Action* specifies the label (identifier) to be used by the observer to denote the requirement's violation. The same label will be used in the reachability graph output by TTool and RTL, in such a way one may easily set up a correspondence.

The metamodel of TURTLE Requirement Diagram is depicted by Figure 4. In TURTLE, a Requirement Diagram (*TRequirement_Diagram* class) extends the SysML Requirement Diagram (*::SysML::Requirement diagram* class). It is made up of three types of nodes: informal requirements (*TInformal_Requirement* class), formal requirements (*TFormal_Requirement* class) and observers (*TObserver* class).

Informal and formal requirements are new stereotypes defined from SysML Requirement (*::SysML::Requirement* class). A formal requirement derives from an informal one (association labelled by *derive*).

TObserver is a new stereotype defined from SysML stereotype "TestCase" (*::TTDTestingProfile:: TestCase* class) which corresponds to the device used for requirement verification. An observer verifies only one formal temporal requirement (represented by the association labelled by *verify* which binds a "TestCase", an observer in this paper, with a formal requirement described by a TRDD).

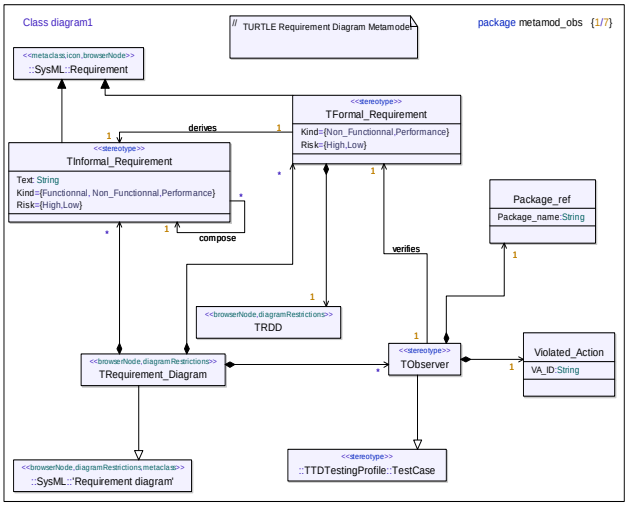

Figure 4: Metamodel of TURTLE Requirement Diagram

3.3 Timing Requirement Description Diagram (TRDD)

A TRDD describes one temporal requirement. The TRDD in Figure 5 refers to a process which must complete within 10 time units. The process is defined by two actions "*Start_Process*" and "*End_Process*" that we call "*observations points"*. The latter are modeled above the TRDD lifeline. This latter includes a temporal frontier (equal to 10 time units in this example). The "temporal frontier" distinguishes between two time periods - OK and KO - that correspond to a requirement satisfaction and violation, respectively.

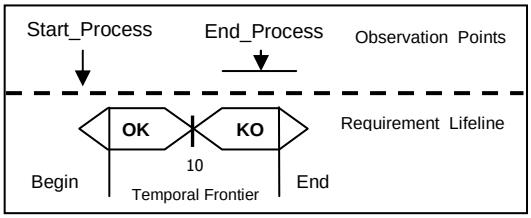

Figure 5: Timing Requirement Description Diagram

Note. Figure 5 uses two comments ("Observation points" and "Requirement Lifeline") which are not part of the metamodel.

The TRDD metamodel is presented in Figure 6. A TRDD (*TRDD* class) extends UML Timing Diagrams [16] (Value Lifeline Timing Diagram). A *TRDD* contains an attribute *n_TRD* whose value equals the number of elements in the TRDD requirement description (OK or KO).

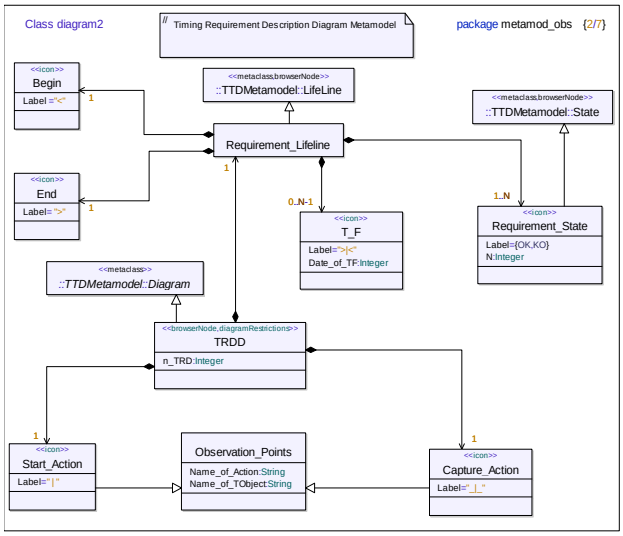

Figure 6: Metamodel of TRDD

As shown in Figure 5, a TRDD is made up of:

- One *Requirement Lifeline* (*Requirement_Lifeline* class, see Figure 6) which contains one *Begin* and *End* symbols, N *Requirement States* (OK or KO) and N-1 *Temporal Frontiers*.

- Two *Observation Points* (*Observation_Points* class, cf. Figure 6) which correspond to those events describing requirement observations. *Start_Action* models the beginning of a requirement capture. *Capture_Action* models the end of the requirement capture.

3.4 Expression power of TRDDs

The purpose of this section is to exemplify which type of Temporal Requirement (TR) may be modelled using a TRDD. Two classes of TRs are identified in [WAH 94]:

- Requirements where time is expressed in a qualitative way. This class of requirements exclusively considers partial order between events.
- Requirements where time is expressed in a quantitative way. This class of requirements considers both the order of events and temporal distances between these events.

This paper focuses on temporal requirements where time is represented in a quantitative way.

This kind of requirement corresponds to *bounded promptness properties* [1] resulting from the class of *safety properties*. This kind of requirement must have a deadline after which a property is not satisfied [1]. This implies that the system must be temporally bounded; otherwise, it cannot be verified.

Table 3 presents temporal requirements corresponding to the bounded promptness properties defined in [1]. The relation which denotes that a system S satisfies a requirement R is written $S \models R$.

Kind of TR	Definition
Promptness	R ensures that an event must occur before a deadline Tmax. $S \models R$ is true if this event occurs before Tmax.
Minimal Delay	R ensures that an event must occur after a minimum time Tmin. $S \models R$ is true if this event occurs after Tmin.
Punctuality	R ensures that an event must occur at one punctual date T. $S \models R$ is true if this event occurs at the T date.
Periodicity	R ensures that an event must occur regularly at modulo T dates. $S \models R$ is true if this event occurs at modulo T dates.
Interval Delay	R ensures that an event must occur between/outside a temporal interval]Tmin; Tmax[. $S \models R$ is true if this event occurs between/outside temporal interval]Tmin; Tmax[.

Table 3: Temporal Requirements taxonomy based on [1]

As shown in Figure 7, different patterns of TRDD are compared with the Temporal Requirements (TRs) presented in Table 3. We distinguish between three classes of TRDD patterns (see Figure 7):

- a) TRDDs with one temporal frontier where corresponding events must occur before/after T time units. These requirements correspond to *Promptness* and *Minimal Delay* requirements, respectively.
- b) TRDDs with two temporal frontiers where corresponding events must occur between/outside the interval *]T1; T2[*. These requirements correspond to *Interval Delay* and *Punctuality* Requirements, respectively. A TRDD with two temporal frontiers T1 = T-1 and T2 = T+1 corresponds to *Punctuality* Requirement on date T. Punctuality is verified on both sides of date T. Note: Integer bounds are supported.
- c) TRDDs with N temporal frontiers and N+1 requirement states (OK or KO), where P1 and P2 represent two possibilities of requirement states (OK or KO). This kind of requirements is not referred in the taxonomy presented in Table 3. Thus, TRDDs make it possible to express requirements not listed in Table 3.

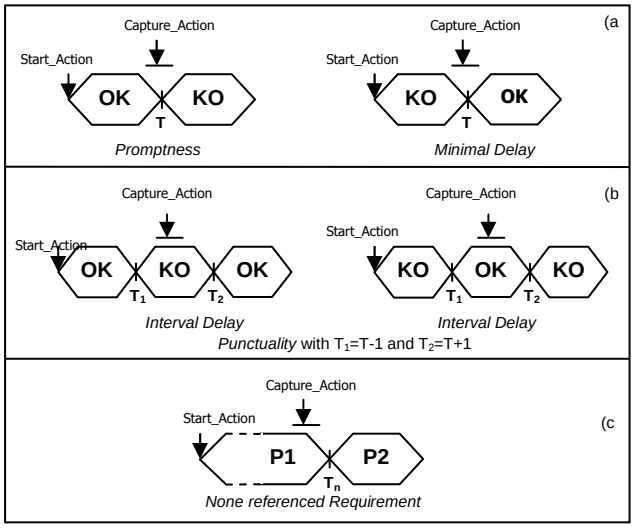

Figure 7: Comparison between TRDD patterns and requirements presented in table 3

All the requirements defined by a TRDD might be periodic (*Periodicity* Requirement). Observers built upon TRDDs may restart on the first point of observation as soon as the second point was met, and if the temporal requirement is satisfied.

4. Case Study: Hybrid Power Management of a Hybrid Vehicle

4.1 Overview

The battery of a hybrid vehicle (HV) is often solicited by the engine for the propulsion and cannot reload permanently. It is thus necessary to reload it when the car produces energy, e.g. during the deceleration or braking phases (represented in Figure 9 by dashed areas). Figure 8 depicts the evolution of

mechanical and electrical power over time; they are depicted by the red and blue curves, respectively.

The scenario depicted in Figures 8 and 9 corresponds to a standard city trip where the driver pushes the pedal accelerator, releases it, then slows down and pushes the brake pedal (e.g. at a red traffic light). This scenario is used in the model presented in 4.3.

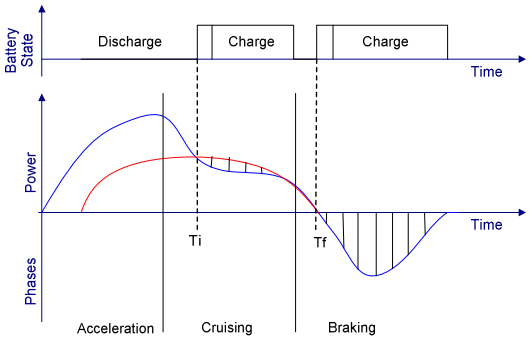

Figure 8: Evolution of mechanical and electrical power during time

Figure 9 presents, on the one hand, the interactions of the pilot with the brake (B) and accelerator (A) pedals (e.g. in Braking phase the driver pushes on the brake pedal) and, on the other hand, the timing diagram which includes dates where the battery must be charged at Ti and Tf dates, with a authorized latency d which corresponds to the system response time.

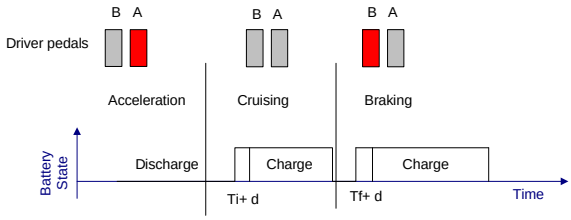

Figure 9: Hybrid Power Management Temporal Requirement

4.2 Requirement Capture

Figures 8 and 9 serve as references for building the Requirement Diagram depicted in Figure 10. Temporal requirements during the Cruising and Braking phases are formalized by the two TRDDs presented in Figure 10.

4.3 Design

Design diagrams include one class diagram (CD) and a set of timed activity diagrams (AD).

Figure 11 shows the class diagram of the hybrid vehicle which contains *Pedal_units* class with the accelerator and brake pedals, *CAN_Bus* class which represents the CAN network, *MicroCont* class is a model of the microcontroller and motor of the HV and *Battery* class which is concerned by the requirements depicted in Figure 10. We intentionally add a model of the driver (*Driver* class) which represents the scenario depicted by Figure 9 (Acceleration, Cruising and Braking).

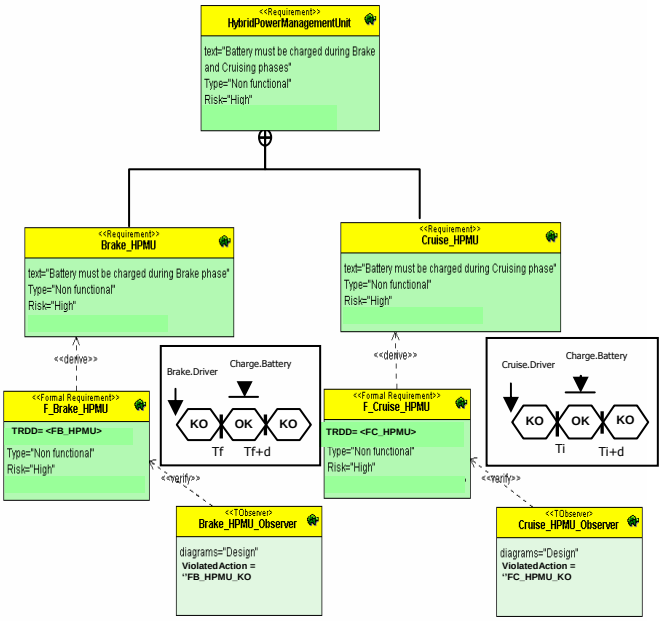

Figure 10: Requirement Diagram and TRDD of HPMU Temporal Requirements

One may observe that the TURTLE Classes are composed by relations attributed with an associative class labeled by “Synchro”. Indeed, TURTLE objects rendezvous in a LOTOS fashion [8], via communication gates described with OCL relations.

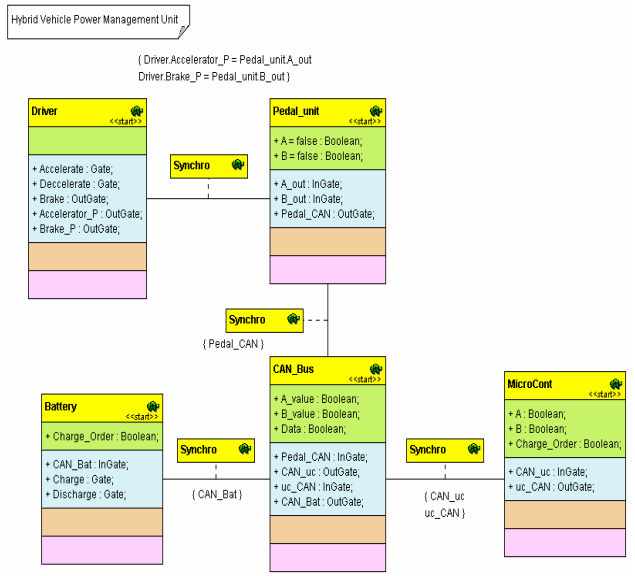

Figure 11: Architecture specified in a TURTLE Class Diagram

4.4 Verification guided by observers

Observers are synthesized from one TRDD to a TIF specification. This specification is composed with the TIF translation of the CD and ADs. Observers and observed objects also communicate by rendezvous. Of interest to us is the "time limited offer" temporal operator which limits the amount of time that may be allocated to offering a rendezvous.

Figure 12 shows the translation process to generate the observer checking for the F_Brake_HPMU requirement. The behavior of the observer is generated starting from the TRDD. First of all, two tables are created to isolate various labels from the TRDD which denotes a requirement satisfaction or violation (see Translation table of TRDD in the observer activity diagram in Figure 12). These tables explain how to build the heart of the observer activity diagram. This embryonic observer's behavior (see Figure 12) is made up of time limited offers, each of them observing one temporal frontier of the TRDD. Additionally, the observer is assembled "upside down".

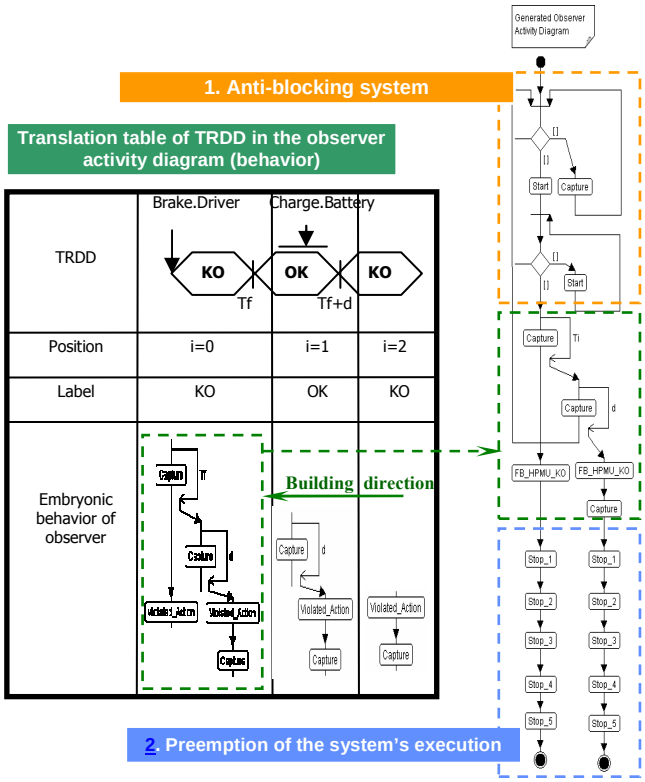

Figure 12: Observers automatic synthesis for the F_Brake_HPMU Requirement

The observer's activity diagram is extended according to the two following rules:

1. The first observation point called "Start" is synchronized with the "Brake" signal of the Driver Class. The observer does not block if two actions "Start" are executed before one action "End" occurs.
2. Since the formal requirement's risk level is set to "high" (see Figure 12), the observer stops the system's execution once the property is not satisfied. The observer executes the actions (*stop_i* actions) that preempt each active class of the system.

Note: Observers synthesis algorithms and metamodels are detailed in [20].

Finally, Figure 13 shows how the results of formal verification of temporal requirements are displayed in a traceability matrix. The latter is automatically built by TTool from the reachability graph. In this example, formal requirements F_Brake_HPMU and F_Cruise_HPMU are violated (see the KO labels in the satisfiability column).

The reachability graph generated from the model with no observer contains 50 states and 60 transitions. Adding one observer leads to distinguish between two situations. If the requirement is satisfied, the graph has 53 states and 63 transitions. Otherwise, the graph has 78 states and 92 transitions. A requirement violation introduces new states in the graph. Additional paths are introduced to characterize the requirement violation and the pre-emption messages.

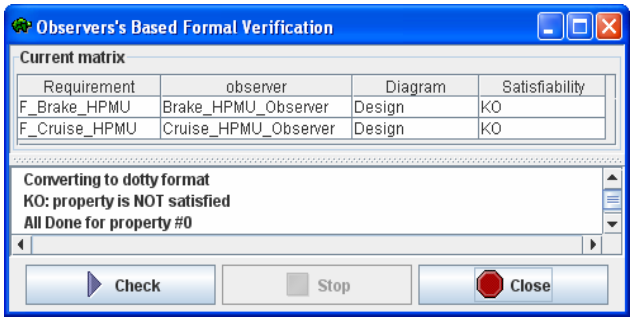

Figure 13: Traceability Matrix of Hybrid Power Management Unit Requirements

5. Conclusions and Future Work

TURTLE is a real-time UML profile designed with formal verification in mind. The profile was recently extended with SysML requirement diagrams. The objective is to formally verify temporal requirements.

The paper shows how SysML requirement diagrams are supported by the profile. Formal temporal requirements are expressed with TRDDs, a graphical language based on UML Timing Diagrams. The paper's contribution lies in the possibility to automatically derive observers from temporal requirements defined by Timing Requirement Description Diagrams. TTool automatically inserts these observers in the relevant design diagrams (class and activity diagrams) as a premise to guide the verification process.

The observer-based verification approach proposed in the paper reuses the RT-LOTOS code generator included in TTool as well as the RTL verification tool. TTool also generates Java code from TURTLE models. We plan to extend the proposed approach to the TURTLE deployment phase of communicating systems. Additionally, observers will be generated in the Java executable code as simulation probes.

6. References

[1] R. Alur and T.A. Henzinger, “Real-time logics: Complexity and expressiveness.” Information and Computation, Volume 104, pp. 35-77, 1993.

[2] N. Amla, E.A. Emerson, and K.S. Namjoshi, “Efficient Decompositional Model Checking for Regular Timing Diagrams”, In Conference on Correct Hardware Design and Verification Methods (CHARME 1999). Springer-Verlag, pp. 67-81, September 1999.

[3] L. Apvrille, J.-P. Courtiat, C. Lohr and P. de Saqui-Sannes, “TURTLE: A Real-Time UML Profile Supported by a Formal Validation Toolkit”, IEEE Trans. on Software Engineering, Volume 30, Number 7, pp. 473-487, July 2004.

[4] L. Apvrille, P. de Saqui-Sannes, R. Pacalet and A. Apvrille, "Un environnement de conception de systèmes distribués basé sur UML", Annals of Telecommunications, Volume. 61, Number 11/12, pp. 1347-1368, November 2006.

[5] V. Braberman, N. Kicillof and A. Alfonso, “A Scenario-Matching Approach to the Description and Model-Checking of Real-Time Properties“, Volume 31, Number 12, pp. 1028-1041, IEEE Transactions on Software Engineering, December 2005.

[6] CADP Website, http://www.inrialpes.fr/vasy/cadp/

[7] H. Chockel and K. Fisler, “Temporal Modalities for Concisely Capturing Timing Diagrams”, Correct hardware design and verification methods, 13th IFIP WG 10.5 advanced research working conference, CHARME’05, Saarbrücken, Germany, October 2005.

[8] J.P. Courtiat, C.A.S. Santos, C. Lohr and B. Outtaj, “Experience with RT-LOTOS, a Temporal Extension of the LOTOS Formal Description Technique”, Computer Communications, Volume 23, Number 12, pp. 1104-1123, 2000.

[9] W. Damm and D. Harel. “LSCs: Breathing Life into Message Sequence Charts”, Formal Methods in Systems Design, Volume 19, Number 1, pp. 45-80, 2001.

[10] M. Fränzle and K. Lüth, “Visual Temporal Logic as Rapid Prototyping Tool”, Computer Languages, Volume 27, pp. 93-113, 2001.

[11] J. Hassine, J. Rilling and R. Dssouli, “Timed Use Case Maps“, In System Analysis and Modeling: Language Profiles, 5th International Workshop, SAM 2006, Kaiserslautern, Germany, pp. 99-114, June 2006.

[12] C. Jard, J.-F. Monin and R. Groz, “Development of Veda, a Prototyping Tool for Distributed Algorithms**,”** IEEE Transactions on Software Engineering, Volume 14, Number 3, pp. 339-352, March 1988.

[13] A. Van Lamsweerde, “Goal-Oriented Requirements Engineering”, System Objectives to UML Models to Software Specifications, Wiley, 2006.

[14] R. Milner, “Communication and Concurrency,” Prentice Hall, 1989.

[15] Objectiver Website, http://www.objectiver.com/

[16] Object Management Group, “Unified Modeling Language Specification”, Version 2.1.1, http://www.omg.org/docs/formal/07-02-03.pdf

[17] RTL Website, http://www.laas.fr/ RT-LOTOS/

[18] Object Management Group, “System Modeling Language Specification”, Version 1.0, http://www.SysML.org/docs/specs/SysML-v1-Draft-06-03-01.pdf

[19] TTool Website, http://labsoc.comelec.enst.fr/turtle/

[20] B. Fontan, P. de Saqui-Sannes et L. Apvrille, « Synthèse d’observateurs à partir d’exigences temporelles », 14ième conférence Langages et Modèles à Objets (LMO 2008), Montréal, Canada, Mars 2008.

[21] M. Hause, F. Thom and A. Moore, “Inside SysML,” IEEE Computing & Control Engineering, pp. 10-15, September 2005.

[22] S. Turki and T. Soriano, “A SysML extension for Bond Graph support,” 5th Int. Conference on Technology and Automation, Thessaloniki, Greece, October 2005.

8. Glossary

AD	Activity Diagram
BGC	Binary-Coded Graphs
CADP	Construction and Analysis of Distributed Processes
CD	Class Diagram
HPMU	Hybrid Power Management Unit
HV	Hybrid Vehicle
IOD	Interaction Overview Diagram
OCL	Object Constraint Language
RD	Requirement Diagram
RTL	Real Time Laboratory
SD	Sequence Diagram
TIF	TURTLE Intermediate Format
TM	Traceability Matrix
TR	Temporal Requirement
TRDD	Timing Requirement Description Diagram
TTool	TURTLE Toolkit
TURTLE	Timed UML and RT-LOTOS Environment
UCD	Uses Cases Diagram